A Parallel Platform for Mining General Co-Movement Patterns over Large-scale Trajectories

ABSTRACT

1. INTRODUCTION

The prevalence of positioning devices has drastically boosted the scale of trajectory data collection. Instances include telemetry chips for animal herding histories, GPS devices for urban transportation and wearable devices for personal moving activities. These positional data are timestamped and can be viewed as trajectories. Data analysis on large-scale trajectories benefits a wide range of applications and services, including traffic planning [1], animal analysis [2], and social recommendations [3], just to name a few.

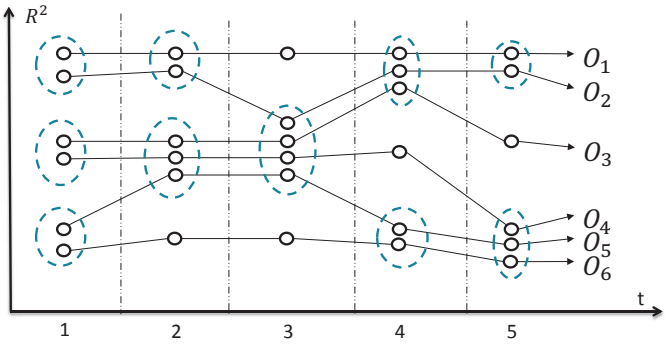
An important task of data analysis on trajectories is to discover co-moving objects. As its name suggests, a *co-movement* pattern [4] consists of a group of objects that travel together for some duration, where the group is formed by certain clustering methods. A pattern is prominent if the size of the group exceeds M and the length of the duration exceeds K, where M and K are user specified parameters. Depends on further constraints on object clustering and time duration, several specific patterns are proposed in the literature.

Specifically, on the aspect of object clustering, *flock* [5] and *group* [6] patterns require objects to be within a disk region. In contrast, *convoy* [7], *swarm* [8] and *platoon* [9] patterns require objects to be densely connected. In terms of the constraints on time duration, *flock* [5] and *convoy* [7] require the timestamps to be consecutive. *Swarm* [8], on the other hand, do not impose any restrictions. *Group* [6] and *platoon* [9] introduce the parameter L to control the minimum length of the local-consecutive timestamps. The compare and contrast of these co-movement patterns are presented in Table 1.

We illustrate these co-movement patterns in Figure 1 with an example of five trajectories. First, the temporal domain is discretized into five snapshots. In each snapshot, the objects are clustered as shown in the dotted circle. By setting the parameters $M = 2, K = 3$, we are able to discover *convoy*, *flock*, and *swarm* patterns. It is notable that *convoy* and *flock* have the same result pattern, this is because both patterns share the same constraints on the time duration. *Swarm* pattern results in a super set of *convoy* and *flock* patterns, as it does not require any consecutiveness on the time duration. By setting $L = 2$, we find three *group* patterns. If further set $M = 2, K = 3$, two *platoon* patterns are found. It is observable that the pattern $\langle\{O_5, O_6\}\{1, 4, 5\}\rangle$ is a *swarm* but is neither a *platoon* nor a *group* since one of its local consecutive duration (i.e. $\{1\}$) is less than 2. However, $\langle\{O_5, O_6\}\{1, 4, 5\}\rangle$ can be reduced to $\langle\{O_5, O_6\}\{4, 5\}\rangle$ which is a *group* pattern because *group* pattern only restricts the local consecutiveness but not the total size of a duration.

Pattern	Clustering Method	Consecutiveness on Duration
Flock [10]	Disk region	Consecutive
Convoy [7]	Density	Consecutive
Group [6]	Disk region	Local consecutive $\geq L$
Swarm [8]	Density	No constraint
Platoon [9]	Density	Local consecutive $\geq L$

Table 1: Existing Co-movement Patterns



Flock(M=2, K=3)	$\{o_3, o_4\}\{1,2,3\}$
Convoy(M=2, K=3)	$\{o_3, o_4\}\{1,2,3\}$
Group(L=2)	$\{o_1, o_2\}\{1,2,4,5\}, \{o_3, o_4, o_5\}\{2,3\}, \{o_5, o_6\}\{4,5\}$
Swarm (M=2,K=3)	$\{o_1, o_2\}\{1,2,4,5\}$, $\{o_5, o_6\}\{1,4,5\}$, $\{o_3, o_4\}\{1,2,3\}$
Platoon (M=2,K=3,L=2)	$\{o_1, o_2\}\{1,2,4,5\}, \{o_3, o_4\}\{1,2,3\}$

Figure 1: Trajectories and Co-Movement Patterns

As can be seen, there are various co-movement patterns requested by different applications and it is cumbersome to design a tailored solution for each type. Hence, there is an urgent demand for a general platform to support all these

co-movement patterns. Another issue with existing methods is that they are built on top of centralized indexes that are not scalable. The maximum number of trajectories ever evaluated is up to hundreds of thousands of trajectories. In practice, it is rather common to collect millions of trajectories. We, for the first time, evaluate the pattern mining performance at the million scale and report the performance in Figure **??**. The results show that their performance degrades dramatically as the dataset size scales up. None of them can handle millions of trajectories efficiently.

Therefore, our primary contributions in this paper are to close these two gaps. First, we propose a general co-movement pattern definition to incorporate all the related variants in the existing literature. We observe that even though *platoon* is the most general pattern among existing patterns, it suffers from *loose connection* problem. This is because that *platoon* allows the timestamps in a pattern duration to be in arbitrary distance, making the object group loosely connected. For instance, patterns with duration $\{1, 2, 100, 101\}$ could be a valid *platoon*; however, the two timestamps $2, 100$ are too far from each other. Such a pattern is likely to be induced by periodic movements of unrelated objects thus is of low interests. To cope with this anomaly, we propose the *general co-movement pattern* by introducing the gap parameter G, which requires the distance of timestamps in the duration to be no larger than G. By so doing, we gain a fine-grained control over the pattern duration, which alleviates the *loose connection* problem. Meanwhile, the general co-movement pattern does not concede its expressiveness. As we show in later sections, the general co-movement pattern is able to reduce to existing patterns by setting appropriate parameters.

Second, we propose a parallel solution based on Spark for scalable pattern mining. Our solution is a three-step MapReduce alike approach. First, trajectories are partitioned into snapshots where clusters from each snapshots are formed in parallel. Second, clusters at various timestamps are shuffled and regrouped. Third, the general co-movement patterns are mined from each group. The challenge here is to ensure that the patterns mined at step three is complete. If not, further shuffles and computations are required which significantly drag down the system performance. In order to resolve the challenges and meanwhile keep the shuffle amount at each stage to a minimum, we design a novel star-based partition scheme to efficiently partition objects based on their belonging clusters. Based on the star partition, we then propose a series of optimization techniques which largely improve the system performance.

Third, we conduct a set of extensive experiments on XXX datasets with million-scale trajectories. The results show that XXX.

The rest of our paper is organized as follows: Section 2 summarizes the relevant literature on trajectory pattern mining; Section 3 forms the definition of the general co-movement pattern mining; Section 4 presents our parallel architecture; The solution of mining the general co-movement pattern mining is presented in Section **??** and Section 6 discuss various optimization techniques to boost the system performance; Section 7 conducts extensive experiments to showcase the usefulness and efficiency of our system and finally Section 8 concludes our paper.

2. RELATED WORKS

The *co-movement patterns* in literature consist of five members, namely *group* [6], *flock* [10], *convoy* [7], *swarm* [8] and *platoon* [9]. We have demonstrated the semantics of these patterns in Table 1 and Figure 1. In this section, we focus on comparing the techniques used in these works. For more trajectory patterns other than *co-movement patterns*, interested readers may move to [11] for a comprehensive survey.

2.1 Flock and Convoy

The difference between *flock* and *convoy* lies in the object clustering methods. In *flock* objects are clustered based on their distance. Specifically, the objects in the same cluster needs to have a pair-wised distance less than *min_dist*. This essentially requires the objects to be within a disk-region of delimiter less than *min_dist*. In contrast, *convoy* cluster the objects using density-based clustering [12]. Technically, *flock* utilizes a m^{th}-order Voronoi diagram [13] to detect whether a subset of object with size greater than m stays in a disk-region. *Convoy* employs a trajectory simplification [14] technique to boost pairwise distance computations in the density-based clustering. After clustering, both *flock* and *convoy* use a line-sweep method to scan each snapshots. During the scan, the object group appears in consecutive timestamps is detected. Meanwhile, the object groups that do not match the consecutive constraint are pruned. However, such a method faces high complexity issues when supporting other patterns. For instance, in *swarm*, the candidate set during the line-sweep grows exponentially, and many candidates can only be pruned after the entire snapshots are scanned.

2.2 Group, Swarm and Platoon

Different from *flock* and *convoy*, all the *group,swarm* and *platoon* patterns have more constraints on the pattern duration. Therefore, their techniques of mining are of the same skeleton. The main idea of mining is to grow object set from an empty set in a depth-first manner. During the growth, various pruning techniques are provided to prune unnecessary branches. *Group* pattern uses the Apriori property among patterns to facilitate the pruning. *Swarm* adapts two more pruning rules called backward pruning and forward pruning. *Platoon* further adapts a prefix table structure to guide the depth-first search. As shown by Li et.al. [9], *platoon* outperforms other two methods in efficiency. However, the three patterns are not able to directly discover the general co-movement pattern. Furthermore, their pruning rules heavily rely on the depth-first search nature, which lost its efficiency in the parallel scenario.

THESE WORKS ARE MOST RELATED TO OUR PROBLEMS, SO I REMOVED OTHER RELATED WORKS FOR NOW.

3. DEFINITIONS

Let $\mathbb{O} = \{o_1, o_2, ..., o_n\}$ denote the set of objects in concern. We use $T \subseteq \{1, 2, ..., m\}$ to denote a descritized sequence of timestamps, where $T[i]$ denotes the i^{th} entry in the sequence and $|T|$ is the cardinality of the sequence.

A time sequence is consecutive if the difference between any of its neighboring entries is 1, that is $\forall T[i], T[i+1] \in T, T[i+1] = T[i] + 1$. For any time sequence, it can be decomposed into at least one consecutive portion(s). Therefore we can define the L-consecutiveness as follows:

Pattern	L	G	M	K
Group	·	$\|T\|$	2	1
Flock	K	1	·	·
Convoy	K	1	·	·
Swarm	1	$\|T\|$	·	·
Platoon	·	$\|T\|$	·	·

Table 2: Representing other patterns using GCMP. · means user specified value.

Definition 1 (L-consecutive [9]). *A time sequence T is L-consecutive if each of its consecutive portions has cardinality greater or equal to L.*

Take $T = \{1, 2, 3, 5, 6\}$ as an example. T is not 3-consecutive since $\{5, 6\}$ has a cardinality less than 3. T is 2-consecutive since both of its consecutive portions are of sizes at least 2. To control the closeness of timestamps, we further define the G-connected of a time sequence as follows:

Definition 2 (G-connected). *A time sequence T is G-connected if the gap between any of its neighboring timestamps is no greater than G. That is $\forall T[i], T[i+1] \in T, T[i+1] - T[i] \leq G$.*

Take $T = \{1, 2, 3, 5, 6\}$ as an example. T is 2-connected since the maximum gap between its neighboring time stamps is $5 - 3 = 2$. However, T is not 1-connected. Indeed, 1-connection infers that T is consecutive.

Given a timestamp t, objects with their locations at t collectively form a *snapshot*[1].Objects in a snapshot can then be clustered based on the closeness of their locations. Let $C_t(o_i)$ be the cluster which o_i belongs to at time t, a general co-movement pattern can be defined as:

Definition 3 (General Co-Movement Pattern). *A* General co-movement pattern *$GCMP(M, K, L, G) = \langle O, T \rangle$ is a pair containing an object set $O \subseteq \mathbb{O}$ and a time sequence T, with the following constraints: (1)Closeness: $\forall o_i, o_j \in O, \forall t \in T, C_t(o_i) = C_t(o_j)$. (2) Significance: $|O| \geq M$. (3) Duration: $|T| \geq K$. (4) Consecutiveness: T is L-consecutive, and (5)Connection: T is G-connected.*

The general co-movement pattern retains the patterns that discovered by all existing techniques (group, flock, convoy, swarm and platoon). The relationships between general co-movement pattern and other patterns are summarized in Table 2.

In particular, by setting $G = |T|$, we achieve the *platoon* pattern. By setting $G = |T|, L = 1$, we achieve the *swarm* pattern. By setting $G = |T|$, $M = 2$, $K = 1$, we gain the *group* pattern. Finally by setting $G = 1$, we achieve the *convoy* and *flock* pattern. In addition to covering existing patterns, the general co-movement pattern avoids the *loose connection* problem in *platoon* pattern. As suggested previously, $\{1, 2, 100, 101\}$ will be included in the platoon pattern, however since they're too far away, this pattern is not prominent. By setting appropriate G, we are able to prune this anomaly. It is notable that GCMP is not able to be modeled by existing patterns.

It is also observable that the number of patterns in GCMP is exponential. To control the size of output, we notice that, for two patterns P_1, P_2, if $P_1.O \subseteq P_2.O$ and P_2 is a proper pattern, then P_1 is also a proper pattern. Therefore, we can define the *Closed General Co-Movement Pattern* as follows:

Definition 4 (Closed General Co-Movement Pattern). *A general co-moving pattern $P = \langle O, T \rangle$ is closed if and only if there does not exist another general co-moving pattern P' s.t. $P.O \subseteq P'.O$.*

For example, let $n = 2, k = 2, l = 1, g = 1$, the pattern $\{o_1, o_2\}\{1, 2, 3, 4\}$ is not a closed pattern, while $\{o_1, o_2, o_3\}$ $\{1, 2, 3, 4\}$ is a closed pattern. The closed pattern avoids outputting duplicate information, thus making the result patterns more compact.

LETS KEEP THE CLOSENESS INFORMATION AT THE MOMENT. IF NO CLOSENESS IS DEFINED, WE CANNOT REDUCE GCMP TO OTHER PATTERNS SINCE THOSE PATTERNS ARE ALL DEFINED AS "CLOSED"

Our definition of GCMP is free from clustering method. Users are able to supply different clustering method to facilitate different needs. We currently expose both disk-region based clustering and DBSCAN as APIs to the user.

In summary, the goal of this paper is to present a parallel solution for discovering closed GCMP from large-scale trajectory data.

Before we move on to the algorithmic part, we list the notations that are used in the following sections.

Symbols	Meanings
Tr_i	Trajectory of object i
S_t	Snapshot of objects at time t
$\mathbb{O}$	Set of objects
T	Time sequence
$C_t(o)$	the cluster of object o at time t
Sr_i	The star structure of object i

Table 3: Notions that will be used

4. OVERVIEW OF MINING GCMP IN PARALLEL

We adapt the MapReduce paradigm for designing a parallel solution of mining GCMP. MapReduce was proposed by Dean et. cl. [] and has become a mature parallel platform for large-scaled data processing. Current open source MapReduce systems provide handy programming APIs with fault tolerances in backends. Such systems include Apache Hadoop, Apache Shark and Apache Spark to name a few.

In simple words, there are two types of cluster nodes in MapReduce, namely the *mapper*s and the *reducer*s. The execution of a MapReduce task consists of three stages: First, input data are partitioned and read by a *map* function on each mapper. Then, mappers emit key-value pairs which are *shuffle*d over the network to reducers. Finally, reducers process received data using a *reduce* function then write the outputs. Since the *shuffle* stage needs to transfer data over network, an important attention to pay during designing MapReduce algorithms is to minimize the shuffle amounts and shuffle counts.

Our GCMP mining workflow follows the MapReduce style which consists of five stages as illustrated in the Figures 2 (a)-(e). As shown, in stage (a), trajectory data are read from various sources and partitioned based on objects timestamps, where objects of the same timestamps form a snapshots. After partitioning, snapshots are send to workers

[1]Missing time stamps can be interpreted using existing methods such as linear interpolation [7].

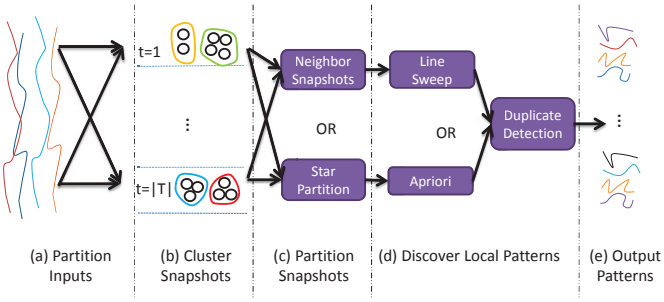


Figure 2: System Flow of Mining GCMP

prior to stage (b). In stage (b), objects belonging to the same snapshots are clustered based on the user supplied clustering method. Then those clusters are shuffled to workers prior to stage (c). In stage (c), each worker collects the clusters sent to its partition and discover the GCMP from these local clusters. A post-check is required to remove duplicate patterns. In stage (d), patterns discovered in previous stages are write to output.

The first two stages of our solution could be straightforward. However, it is challenging to design the partition and mining methods to in stage (c) and (d). Intuitively, A reasonable partition strategy needs to meet the following requirements: first, the result partitions need to preserve enough information so that real patterns can be discovered in stage (d). Second, the result partitions need to ensure that the patterns that discovered in stage (d) are real patterns so that no further validation is required after stage (d). We formalize the two properties using *completeness* and *soundness* as follows:

Definition 5 (Completeness and Soundness)**.** *Let a partition method $\mathbb{P}$ partitions original trajectories Tr into multiple parts, $Par_1, ..., Par_m$. $\mathbb{P}$ is complete if for every pattern P that is valid in Tr, $\exists Par_i$ such that P is valid in Par_i. $\mathbb{P}$ is sound if for all patterns P that is valid in any Par_i, it is also valid in TR.*

The completeness ensures that no real patterns are missed out. The soundness ensures that no false patterns are reported. If a partition method is both sound and complete, we are then able to mining patterns in each resulted partition in parallel in stage (d).

Apparently replicating entire data set to each partition meets the *soundness* and *completeness*, however, it burdens the network shuffle and GCMP mining. Our objective is thus to design a complete and sound partition method that minimize the network shuffles.

5. TEMPORAL REPLICATION AND MINING

The straightforward strategy of parallelizing the second stage is to vertically partition the trajectories based on snapshots. A simple but effective method is to group neighborhood snapshots into a partition, such that every possible pattern can be mined within some of the partitions. In order to achieve the *completeness*, some of the snapshots need to be replicated on multiple executors. We call this method the *Temporal Replication and Mining* approach. The algorithm is presented as in Algorithm 1.

Algorithm 1 Temporal Replication and Mining

Require: list of $\langle t, S_t \rangle$ pairs
  Map Phase
  for all $\langle t, S_t \rangle$ **do**
    for all $i \in 1...(K-1)*G+K$ **do**
      emit a $\langle t-i, S_t \rangle$ pair
    end for
  end for
  Shuffle Phase
  for all $\langle t, S \rangle$ pair **do**
    group-by t, emit a $\langle t, Par_t \rangle$,
    where $Par_t = \{S_t, S_{t+1}, ..S_{t+(K-1)*G+K}\}$
  end for
  Reduce Phase
  for all $\langle t, Par_t \rangle$ **do**
    minePattern(Par_t)
  end for

5.0.1 Temporal Replication Partition

In order to reduce the shuffling cost, the data to be replicated should be kept to a minimum. However, as suggested in the following theorem, the minimum replication of a snapshot is not small:

Theorem 1 (Soundness and Completeness of Replication)**.** *For each snapshot S_t, a partition $Par_t = \{S_t, ..., S_{t+(\lfloor \frac{K}{L}-1 \rfloor * G+K)}\}$ is created. Such a partition method is sound and complete.*

Proof. The soundness of partition is trivially true by definition. Given a valid pattern P, let $T' \subseteq P.T$ be a subsequence of $P.T$ which conforms to K, L, G. Note that there could be many qualified T' of T. Then, let the i^{th} local-consecutive part of T' be l_i and let the i^{th} gap of T' be g_i. Then, the size of T' can be written as $\Sigma_i(l_i + g_i)$. Since T' conforms to K, L, G, then $\Sigma_i(l_i) \geq K$, and $K \geq l_i \geq L$, $g_i \leq G$. Therefore, among all possible T's, the minimum size is $\lfloor \frac{K}{L} - 1 \rfloor * G + K$. Thus ensuring each Par_t to be of that size would capture one of the T's, therefore the pattern P is valid in Par_t. This proves the completeness of the partitioning method. □

Utilizing Theorem 1, we create, for each snapshot S_t, a partition containing its next $\lfloor \frac{K}{L} - 1 \rfloor * G + K$ snapshots. Each partition is then sent to the executors as a task. Since any global pattern must exists in one of the partitions, we can mine the patterns from each partitions independently, without loss of patterns.

5.0.2 Temporal Replication Mining

After replication, each task in Spark will process an partition Par_i. The next step is to mine the GCMPs from Par_i. We notice that, for each partition, we only need to find the patterns that are contained in the first snapshots. Therefore we design a line-sweep method for mining such patterns, which is the variant of the Coherent-Moving-Clustering method in [7].

The Temporal Replication and Mining approach though achieves parallelism from independent partitions, it requires to replicate the data multiple times. Specifically, each snapshots are copied $(K-1)*G+K$ times. In swarm case, G is as large as $|\mathbb{T}|$. In such a case, it is equivalent to replicate the entire dataset to each executor, which is clearly inefficient.

5.1 Star Partition and Mining

To develop a method that achieves parallelism under any pattern parameters, we propose the *Star Partition and Mining* (SPM) method. In SPM, we design a novel object-based partition method named star partition. A start partition partitions trajectories in the object domain rather than temporal domain. After partitioning, we design the *Apriori*-like method to mine the patterns out of each partition independently. The overview of the star partition and mining is as in Algorithm 2.

Algorithm 2 Star Partition and Mining

Require: list of $\langle t, S_t \rangle$ pairs
  Map phase
  for all $C \in S_t$ **do**
    for all $(o_1, o_2) \in C \times C$ **do**
      emit a $\langle o_1, o_2, \{t\} \rangle$ triplet
    end for
  end for
  Shuffle phase
  for all $\langle o_1, o_2, \{t\} \rangle$ triplets **do**
    group-by o_1, emit $\langle o_1, Sr_{o_1} \rangle$
    group-by o_2, emit $\langle o_2, Sr_{o_2} \rangle$
  end for
  Reduce phase
  for all $\langle o, Sr_o \rangle$ **do**
    Apriori(Sr_o)
  end for

5.1.1 Star Partition

The purpose of star partition is to find the group of objects that could potentially form a pattern. In order to achieve parallelism, we group, for each object o, all other objects that connect to o in some snapshots. By so doing, the patterns containing o can be discovered within each groups. Technically, we create a connection graph to represent the connectivity among objects. A connection graph is an undirected graph $G = (V : E)$, where each $v \in V$ represents an object. An edge $e(s, t) = ET$ contains all the timestamps where s, t are in the same cluster, i.e., $\forall t \in ET, C_t(s) = C_t(t)$. Given a vertex s, the star of s, Sr_s is the set of incidental edges of s in G. An example of star partition is shown in Figure 3.

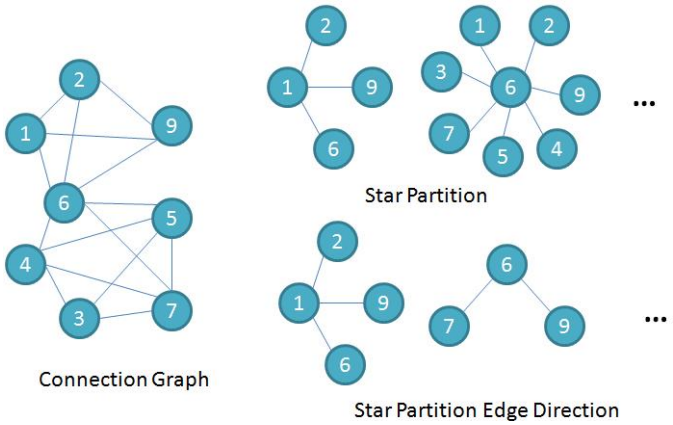


Figure 3: Example of Star Partition

In star partition, trajectories are partitioned based on objects and their stars. Then each partition is send to an executor as a task. Indeed, a star Sr_s can be viewed as a subset of original trajectories. This is done by treating each vertex in Sr_s as an object. For s, the time sequence is the union of all edges in Sr_s. For vertex $v \neq s$, the time sequence is the edge (s, v). Then, we state the completeness of star-partition as in the following theorem:

Theorem 2 (Soundness and Completeness of Star Partition)**.** *Star partition is sound and complete.*

Proof. For the soundness, if P is a valid pattern in Sr_s, then for every object $v \neq s, v \in P.O$, and for every timestamp $t \in P.T$, $C_t(v) = C_t(s)$. This means, for every snapshot t, all object in $P.O$ belongs to the same cluster. Thus, P is a valid pattern in original trajectories.

For the completeness, if P is a valid pattern in original trajectories, for an arbitrary object $s \in P.O$, it follows $P.O \equiv S_r$. Since for any time t, $C_s(t) = C_v(t) \forall v \in P.O$, every time in $P.T$ appears in edges in Sr_s, thus P is valid pattern in Sr_s. □

Based on the above theorem, each star can be processed independently by executors, which avoids the inter-executor communication during the mining phase. It is notable that, the replication of data is $O(|\mathbb{O}|^2|\mathbb{T}|)$, which is free from the parameters of patterns. In later sections, we will describe optimization techniques to reduce the replications.

5.1.2 Apriori Mining

For each task, in the mining phase, we need to find the patterns that conform to the parameters. To systematically discover the patterns, we design the *Apriori Mining* method which is similar to frequent item mining. During the algorithm, we call a candidate pattern R-pattern if the size of its object set is $|R|$. Our algorithm runs in iterations. During each iteration R, we try to generate all $(R+1)$-patterns. In iteration 1, the 2-pattern is the edges in Sr_s. In particular, for each $e(s, v) = ET$, pattern $p = (\{s, v\}, ET)$ is formed. During each iteration, we generate $(R+1)$-cluster patterns by joining R-cluster patterns with 2-cluster patterns. Specifically, the join between $p_1 = (O_1, T_1)$ and $p_2 = (O_2, T_2)$ would generate a new pattern $p_3 = (O_1 \cup O_2, T_1 \cap T_2)$. Notice that in Sr_s, each R-pattern consists of the object s, thus the join will grow a R-cluster at most to a $(R+1)$-cluster. Our mining algorithm stops where no further patterns are generated. The algorithm is illustrated as in Algorithm 3.

Algorithm 3 takes exponential complexity to mine GCMP. There are two major factors dragging the performance. First, the size of Sr_s affects the initial size of 2-patterns. Second, the candidates generated in each level affects the join performance. In later sections, we exploit the property of GCMP to reduce the two factors.

6. OPTIMIZATION

We have analyzed that the bottlenecks of Algorithm 3 lies in two factors. The size of each Sr_s and the size of candidates in each level of Apriori. In this section, we provide several optimizations to boost the bottlenecks.

6.1 Edge Reduction by Direction

The first spot for reducing the size of Sr_s is to remove the replicated edges. As shown in Algorithm 2, each edge

Algorithm 3 Apriori Mining

```
Require: Sr_s
  Lv ← {}
  Ground ← {}
  for all e(s,t) = T ∈ Sr_s do
    Ground.add(⟨{s,t}, T⟩);
    Lv ← Ground;
  end for
  while true do
    if Lv is not empty then
      LvCand ← {}
      for all cand_v ∈ Lv do
        for all cand ∈Ground do
          p ← cand_v join cand
          if p.T is partly valid then
            LvCand.add(p)
          end if
        end for
      end for
      Lv ← LvCand
    else
      break
    end if
  end while
```

in the conceptual graph is replicated twice in generating the star-structure. The purpose of replication is to ensure the completeness of star partition. However, this replication can be avoided if we choose an appropriate way of partitioning.

We design the edge partitioning method by edge direction. Instead of building a conceptual graph that is undirected, we create the directed conceptual graph as follows: First, we assign each object a unique number. Then for a cluster C_t in snapshot S_t, for any pair $(u, v) \in C_t$, an edge $e(u, v) = \{t\}$ is created if $u < v$. It is easy to see that the directed conceptual graph is a DAG. We then create each Sr_u by including all the outgoing edges of u. By so doing, each edge is assigned to only one star, thus avoids the replications. We use the following theorem to ensure the completeness of the edge direction method.

Theorem 3 (Sound and Completeness of Edge Direction)**.** *Star partition with edge direction is sound and complete.*

Proof. It is notable that each star is a subset of original trajectories, thus the soundness is trivially true. For completeness, if P is valid pattern, then let s be the object of the smallest number in $P.O$, i.e., $s = \min_{o \in P.O}(o)$. Since s is smallest and the all other objects in $P.O$ is connected with s. Therefore, $P.O \equiv Sr_s$, which indicates that $P.O$ is also a pattern in Sr_s. □

An example of edge direction is shown in Figure 3. As shown, by adapting the direction method, half the size of S_r's is reduced. This clearly brings efficiency in both shuffling and apriori mining.

6.2 Edge Simplification

Each edge $e(s, t)$ in Sr_s contains a time sequence ET which represent the co-occurrence of s and t. We notice that the edge between s and t is not always necessary. For example, if an edge has a cardinality less than K, it is unnecessary to include this edge to Sr_s since it cannot contribute to any patterns. This motivates us to simplify the edges in Sr_s to boost the overall performance.

We first define the *Pseudo-consecutiveness* of a time sequence as follows:

Definition 6 (Pseudo-consecutiveness)**.** *Given a parameter G, a sequence T is* pseudo-consecutive *if and only if for any $i \in [1, |T|]$, $T[i] - T[i-1] \leq G$*

For example, let $G = 2$, then the sequence $T_1 = \{1, 2, 4, 5\}$ is pseudo-consecutive while $T_2 = \{1, 4, 5, 6, 8\}$ is not pseudo-consecutive.

A pseudo-consecutive subsequence of T is a subsequence $T' \subseteq T$ and T' is pseudo-consecutive. A maximal pseudo-consecutive subsequence T^m of T is a pseudo-consecutive subsequence of T and there is no superset of T^m that is also a pseudo-consecutive subsequence of T. Clearly, a sequence T can be decomposed into several maximally pseudo-consecutive subsequences. We then define a *partly candidate* sequence as follows:

Definition 7 (Partly Candidate Sequence)**.** *Given the pattern parameters: L, K, G, a sequence T is a* partly candidate *sequence if exists one of its maximal pseudo-consecutive subsequence T' such that T' confirms to L, K, G.*

For example, let $L = 2, K = 4, G = 2$, sequence $T_1 = (1, 2, 4, 5, 6, 9, 10, 11)$ is a *partly candidate sequence* since $T_1[1 : 5] = (1, 2, 4, 5, 6)$ is a valid pattern wrt. L, K, G. In contrast, $T_2 = (1, 2, 5, 6, 7)$ is not a valid partly candidate sequence.

Observing that only partly candidate sequence can be potentially contribute to a pattern. Therefore, given an edge $e(s, t) = T \in Sr_s$, if T is not a partly candidate sequence, it can be pruned from Sr_s. To efficiently test whether a given sequence is partly candidate, we define the *Fully Candidate Sequence*:

Definition 8 (Fully Candidate Sequence)**.** *Given the pattern parameters: L, K, G, a sequence T is a* Fully Candidate *if and only if for any of its maximal pseudo-consecutive sequence T', T' is partly consecutive.*

For example, let $L = 2, K = 4, G = 2$, sequence $T_1 = (1, 2, 4, 5, 6, 9, 10, 11)$ is not a fully candidate sequence since one of its maximal pseudo-consecutive sequence $(9, 10, 11)$ is not a partly candidate sequence. In contrast, sequence $T_2 = (1, 2, 4, 5, 6)$ is a fully candidate sequence.

Based on the fully candidate sequence, we can reduce an sequence T to a fully candidate sequence by striping out its non-partly candidate maximal pseudo-consecutive sequences. The reduction works as in Algorithm 4. It takes two rounds of scan of an input T. In the first round of scan, the consecutive portion of T with size less than L is removed. In the second round of scan, the pseudo-consecutive portion of T with size less than K is removed. Clearly the simplification algorithm runs in $O(|T|)$ time.

We use the following theorem to state the completeness and correctness of our edge reduction algorithm.

Theorem 4 (Soundness and Completeness Edge Simplification)**.** *Star partition with edge simplification is sound and complete.*

Proof. Soundness of the star partition is not affected by edge simplification since each star is a subset of original trajectory. For completeness, consider a sequence T, let T_1 be

Algorithm 4 Edge Simplification

Require: T
  Remove the consecutive portion with size less than L
  $c \leftarrow 0$
  for $i \in (0, ..., |T|)$ **do**
    if $T[i] - T[i-1]! = 1$ **then**
      if $i - c < L$ **then**
        T remove $[c : i)$
      end if
      $c \leftarrow i$
    end if
  end for
  Remove the pseduo-consecutive portion with size less than K
  $s \leftarrow 1$, $c \leftarrow 0$
  for $i \in (0 : |T|)$ **do**
    if $T[i] - T[i-1] > G$ **then**
      if $s < K$ **then**
        T remove $[c : i)$
      end if
      $c \leftarrow i$
      $s \leftarrow 1$
    else
      $s++$
    end if
  end for

the result of T after first round of simplification. By the nature of the first round of simplification, if $P.T \subseteq T$, then $P.T \subseteq T_1$. In the second round of simplification, only the pseudoconsecutive parts with size less than K are removed. This means, the removed parts is not able to contribute to any patterns. Thus if $P.T \subseteq T$, then $P.T \subseteq T'$. □

By leveraging the edge simplification, the size of Sr_s can be greatly reduced. If an edge cannot be reduced to a full candidate sequence, then it is directly removed from Sr_s. If an edge can be reduced to a full candidate sequence, replacing itself by the full candidate sequence results in a more compact storage.

6.3 Pruning Apriori Mining

During the apriori phase, we repeatedly join candidate patterns in different levels to generate a larger set of a patterns. We notice that, such joins can be early terminated by utilizing the monotonic property of GCMP. The intuition is that when we are trying to store a candidate, if its temporal sequence is not a *partly candidate* sequence, then it cannot generate full candidate and thus it can be pruned. This *monotonic property* is explicitly described as in the follow theorem:

Theorem 5 (Monotonic Property of GCMP)**.** *Given the temporal parameters L, G, K, for a candidate cand in Algorithm 3, if cand.T cannot be reduced to a partly candidate sequence, then cand can be pruned.*

Proof. Let $cand_i$ be a i-pattern. If $cand_i$ is generated from $cand_j$ where $j < i$, then if $cand_j.T \subseteq cand_i.T$. That is as the level goes up, the time sequence set of a candidate shrinks. Therefore, if $cand_j.T$ is not a partly candidate sequence, $cand_i.T$ cannot be a partly candidate sequence. □

With the help of the *Monotonic Property*, the number of new candidates in each level is greatly reduced. We verify this in the experiment session as well.

7. EXPERIMENTAL STUDY

8. CONCLUSION AND FUTURE WORK

9. REFERENCES

[1] Y. Zheng, Y. Liu, J. Yuan, and X. Xie, "Urban computing with taxicabs," in *Proceedings of the 13th international conference on Ubiquitous computing*, pp. 89–98, ACM, 2011.

[2] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye, "Mining periodic behaviors for moving objects," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1099–1108, ACM, 2010.

[3] J. Bao, Y. Zheng, D. Wilkie, and M. F. Mokbel, "A survey on recommendations in locationbased social networks. submitted to," *Geoinformatica*, 2013.

[4] X. Li, *Managing moving objects and their trajectories*. PhD thesis, National University of Singapore, 2013.

[5] J. Gudmundsson and M. van Kreveld, "Computing longest duration flocks in trajectory data," in *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, pp. 35–42, ACM, 2006.

[6] Y. Wang, E.-P. Lim, and S.-Y. Hwang, "Efficient mining of group patterns from user movement data," *Data & Knowledge Engineering*, vol. 57, no. 3, pp. 240–282, 2006.

[7] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, "Discovery of convoys in trajectory databases," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 1068–1080, 2008.

[8] Z. Li, B. Ding, J. Han, and R. Kays, "Swarm: Mining relaxed temporal moving object clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 723–734, 2010.

[9] Y. Li, J. Bailey, and L. Kulik, "Efficient mining of platoon patterns in trajectory databases," *Data & Knowledge Engineering*, 2015.

[10] J. Gudmundsson, M. van Kreveld, and B. Speckmann, "Efficient detection of motion patterns in spatio-temporal data sets," in *Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pp. 250–257, ACM, 2004.

[11] Y. Zheng, "Trajectory data mining: an overview," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 6, no. 3, p. 29, 2015.

[12] D. Birant and A. Kut, "St-dbscan: An algorithm for clustering spatial–temporal data," *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 208–221, 2007.

[13] P. Laube, M. van Kreveld, and S. Imfeld, "Finding remodetecting relative motion patterns in geospatial lifelines," in *Developments in spatial data handling*, pp. 201–215, Springer, 2005.

[14] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973. doi:10.3138/FM57-6770-U75U-7727.